**Product Classification Models**

**James Lowrey.46, Spencer Morris.911, Alane Suhr.8, David Wright.1325**

**CSE 5522 Final Report**

This semester, our group decided to address a classic problem in artificial intelligence: the classification of objects in a large data set into distinct groups based on multiple distinguishing features. Specifically, an online contest put on by Otto Group and hosted on Kaggle was chosen as a good example of product classification. The contest specifications can be seen at Kaggle’s web site: [Click Here](https://www.kaggle.com/c/otto-group-product-classification-challenge). In this case, the products are characterized by 93 different features whose values are integers from 0 to about 350. The training data set has 61,878 labeled data points. To best classify the products, we chose two different models and implemented them as best we could, then compared the results to an off-the-shelf software package from Scikit-Learn.

**Support Vector Machine**

The model we chose to use as a baseline for product classification was a support vector machine (SVM). SVM’s are supervised learning models that take a geometric approach to learning. The *n* data features are treated as coordinates in an *n*-dimensional space, and the goal is to divide that space into different categories such that the total distance between points and dividing hyperplanes is maximized. Then for prediction of new data, the points can be located in this divided space and their categories can be predicted based on which section they fall in. To test the classification of products using an SVM, our group used Scikit-Learn, which is an open source Python module that offers implementations of common machine learning models. Its installation information and documentation can be found here:<http://scikit-learn.org/stable/index.html>. Scikit-Learn’s implementation of SVM offers multiple concrete classes that vary the internal operations used for learning and prediction while working with the same API. Therefore, we tested various combinations of SVM classes and parameters to find the best results.

The quickest and least effective implementation was LinearSVC(), which uses a linear kernel. It only required about 10 minutes of computation, but had a 74.9% success rate. The next best implementation was SVC() with default parameters, which learns using a radial basis function (RBF) as its kernel. This method required about 30 minutes to run and correctly classified 86.7% of the data. Finally, the last concrete class to choose from was NuSVC(), which offered the most options for configuring the parameters. The best parameters we found in our testing were nu = 0.02, which is an upper bound on the fraction of training errors, kernel = ‘rbf’, degree = 6, gamma = 0.0, coef0 = 0.0, and probability = True. The rest of the parameters were left as defaults. In this configuration, the SVM ran for 35 minutes and had a success rate of 98.6%, by far the best classification rate of the Scikit-Learn SVM implementations. This result from off-the-shelf software was then used as the baseline for comparison of our hand-written models.

**Expectation-Maximization**

Another approach was to implement expectation-maximization on a multinomial mixture model, an unsupervised learning approach. This model learns from *n* examples with *m* features each, attempting to cluster them correctly into *c* clusters. Each feature represented a multinomial distribution of up to *f* possible values, i.e. a given example, for a given feature, will have one of the *f* possible feature values.

Because EM was implemented for Gaussian mixture models earlier in the course, we modified it to learn multinomial distributions rather than Gaussian distributions. This choice was made because for some features, the possible discrete values may be incommensurable, and therefore not representable by a Gaussian mixture model. For example, say a feature represents color and its possible values are red (1), orange (2), yellow (3), green (4), and blue (5). If we chose to attempt to order these values and used a Gaussian mixture model, if a clusters of examples has many examples with value 1 (red) and many with value 5 (blue) (but none which are orange, yellow, or green), the Gaussian distribution would only be able to describe the cluster as having a mean of value 3 (yellow) with a large variance, throwing out the fact that its examples are evenly split along red and blue and it contains no yellow examples. Additionally, because the feature values are discrete, a Gaussian distribution whose parameters are real doesn’t make sense in the context of our data.

We implemented EM in Matlab. Unfortunately, EM did not produce many meaningful results. First, after implementing EM for the multinomial mixture models, we implemented checking how correct our cluster assignments were by comparing it to the original cluster assignments in the original data set. However, this caused runtime errors with Matlab where Matlab’s interpreter would run out of memory. Additionally, when calculating the log probabilities for each cluster-example pair, the disparity between the precision of the log-likelihood of each cluster and the log-likelihoods of each example-cluster and each example caused there to be no real difference between the log probabilities of each cluster-example pair. I.e., because , and because was a large number with low precision and was a very small number, summing the two lost the precision, and therefore the difference, of what would be meaningful in distinguishing the log probabilities of each cluster-example pair. In almost every case, the log probabilities of each cluster-example pair were the same, so clusters were randomly chosen for each example. Because of this, our predicted success for EM would be .

Modifications could be made to improve EM. For example, calculating the parameters in the maximization step could be improved, as currently we are just assigning each example to its most likely cluster and determining parameters from what has been assigned to each cluster. We could also find a way to lower the weight of the log priors in the expectation step so that the difference in and is enough to be meaningful. Another reason EM might not have worked properly is that it makes independence assumptions about each feature, which might not necessarily be true.

**Multinomial Naive Bayes**

Naive Bayes classification is a type of supervised learning that assumes independence between features and uses Bayes theorem to predict the classifications of data. This classifier is one of the most simplistic methods of classification and is commonly used in the language processing field. Essentially, features are tallied in the training dataset and the maximum-likelihood estimation of a test point is found. Smoothing is also applied to prevent a test point from having a zero probability (which would occur if one of its features was not present in the training model). By assuming independence, maximum-likelihood is simplified and general inference is not required, making naive bayes easily scalable. Unfortunately, it is usually not as accurate as more advanced classifiers, such as SVMs.

In this project a Multinomial Naive Bayes was implemented for the Otto Product dataset. To test, the training data was used as both train and test (since the classifications are known), and the project resulted in a 65% accuracy. Laplacian smoothing improved the accuracy from around 62% (no smoothing) to the final 65%. This implementation was written in Java and used the log maximum-likelihood to prevent floating point underflow. It is a very quick test that completes in under a minute with an i5 processor; however, it could easily be further optimized. Lists were used to keep track of ids, as it was theorized that this approach might be more extensible. By changing these lists to an integer variable, processing power could be further saved. Overall, this classifier ran very quickly, but suffered from relatively poor results. The final result of 65% accuracy is not nearly as good as SVM’s, and thus would likely not be applicable in an industry setting. Due to SVMs significant superiority, we can conclude the features are highly unlikely to be independent of one another.

**Future Directions**

Across our different models, the same data used for training was reused for testing. This was done because the training data was the only set available with the correct category included, so it was the only set we could compare for successful categorization. A major downside of this methodology is the risk of overfitting. In future iterations of the project, this problem could be mitigated by splitting the available data into a training set and testing set, which would result in more robust classification. Furthermore, we could evaluate our implementations to classify various other kinds of data, in addition to that provided for the Otto Group competition.